1. Die Idee hinter dem Self-Healing-System

Viele Fehler in unseren Home-Assistant-Protokollen bleiben tagelang unbemerkt. Denn wer schaut schon täglich in die Logs, um wirklich alles im Blick zu behalten? Aus Bequemlichkeit – es muss einfach laufen – verstehe ich das absolut. Ich selbst mache es nicht anders. Doch dabei haben wir immer ein Problem: Was ist, wenn wirklich mal etwas schiefläuft? Dann besteht das Risiko, dass wir die Kontrolle über unser System verlieren. Denn Fehler gibt es nicht grundlos.

Dabei müssen wir beachten, dass es unterschiedliche Einträge in den Protokollen gibt. Wir unterscheiden Warnings von Errors, dazu kommen noch Debug-Einträge – je nachdem, wie viel Einblick das System uns gewährt. In der Regel gilt: Errors haben eine höhere Priorität als Warnings. Das liegt in der Natur der Sache und ist genau das Vorgehen, nach dem sich auch Entwickler richten.

Werfen wir also einen Blick darauf, wie ein Self-Healing-System für Home Assistant in der Praxis aussehen kann – und wie man es idealerweise aufbaut. Bei mir sieht es inzwischen so aus: Ein Fehler wird festgestellt, ein Issue dazu eröffnet, ich schaue kurz drüber und die KI repariert den Fehler anschließend fast komplett von allein.

Die Grundidee stammt aus zwei Quellen. Zum einen habe ich schon länger meine KI-CLI auf die Fehler im Home Assistant losgelassen, um sie zu analysieren und Lösungen zu erarbeiten. Zum anderen habe ich mit Daniel (aka The Smart Home Maker) in einer Podcast-Folge über KI-Anwendungsfälle im Smart Home gesprochen. Aus seiner Idee, die KI gezielt zur Problemlösung einzusetzen, ist dieser Workflow entstanden.

Der Kern des Systems: Fehler und Warnings werden aus dem Protokoll extrahiert und eingeordnet. Eine wichtige Basis bilden die Log-Level Error und Warning. Ebenso spielt ein eigens erzeugter „Fingerprint“ eine Rolle, der Duplikate zusammenfasst. Je nach Fehler wird der Eintrag entweder verworfen oder zur Analyse an die KI gegeben. Der Fehler samt Analyse landet als Issue auf GitHub. Von dort kann ich ihn zum Bugfix freigeben – oder mich selbst darum kümmern.

> **Zwei Stufen mit Mensch in der Mitte**
> Bewusst ist das System zweistufig aufgebaut. Zuerst erfolgt nur eine Analyse, erst danach – und nur nach meiner Freigabe – der eigentliche Fix. Manche Fehler will ich gar nicht von der KI bearbeiten lassen, andere kann sie schlicht nicht lösen. Ist etwa an einem Sensor die Batterie leer, kann keine KI einen Austausch vornehmen. Genau deshalb bleibt die Entscheidung, was passiert, immer bei mir.

2. Architektur im Überblick

Bevor wir Node für Node in die Umsetzung gehen, schauen wir uns die Architektur aus der Vogelperspektive an. Wenn du dieses Bild im Kopf hast, ergibt später jeder einzelne Schritt Sinn.

2.1 Die vier Bausteine

Um das Vorhaben umzusetzen, brauche ich zunächst einen Orchestrator. Das ist eine Software, welche die Koordination des gesamten Ablaufs übernimmt und für die Steuerung zuständig ist. In meinem Fall übernimmt diese Rolle n8n. Rund um n8n gruppieren sich drei weitere Bausteine:

- **n8n (Orchestrator):** Stößt die Abläufe zeitgesteuert an, holt die Logs, bereitet sie auf, ruft die KI auf und schreibt die Ergebnisse nach GitHub. n8n kann bequem auf Home Assistant, GitHub und OpenCode zugreifen, mehrere Auslöser hinterlegen und den Workflow bei Bedarf abschalten.
- **Home Assistant (Datenquelle):** Liefert die Protokolle. n8n holt sie über eine SSH-Verbindung mit dem Befehl ha core logs. Die SSH-Variante deshalb, weil es keine komfortable, stabile API gibt, die die Core-Logs zeilenweise so ausliefert, wie wir sie hier brauchen.
- **OpenCode + Home-Assistant-MCP (die Intelligenz):** Hier steckt die eigentliche Analyse- und Reparatur-Fähigkeit. Über SSH startet n8n eine OpenCode-Session, die per MCP (Model Context Protocol) Zugriff auf Home Assistant hat. Sie kann Protokolle abrufen, den Fehler bestätigen, Kontext sammeln, einen Lösungsweg vorschlagen und – in der zweiten Stufe – auch echte Änderungen vornehmen.
- **GitHub (Schaltzentrale):** Pro Fehler entsteht ein Issue in einem privaten Repository. Es enthält die Fehlerbeschreibung, die KI-Analyse und Meta-Informationen zur OpenCode-Session. Über Labels steuere ich den kompletten Lebenszyklus und kann jederzeit eigene Kommentare hinzufügen.

Optional kommt noch Telegram hinzu: Sobald neue Issues entstanden sind, schickt mir n8n eine kurze Nachricht.

2.2 Warum zwei getrennte Workflows?

Das gesamte System besteht bewusst aus zwei getrennten n8n-Workflows. Der erste – der Log Analyzer – kümmert sich ausschließlich darum, aus Logzeilen saubere, deduplizierte und analysierte GitHub-Issues zu machen. Der zweite – der Self-Healer – nimmt sich später nur jene Issues vor, die ich per Label ausdrücklich zur Reparatur freigegeben habe.

Diese Trennung ist der Dreh- und Angelpunkt des „Mensch in der Mitte“. Zwischen Analyse (Workflow 1) und Fix (Workflow 2) sitzt bewusst meine Entscheidung. So kann eine reine Analyse immer laufen, während die potenziell riskante Reparatur nur auf mein explizites Signal hin startet.

2.3 Der Datenfluss von der Logzeile zum Fix

Kurzum: n8n stößt den Prozess an und holt alle Fehler aus Home Assistant. Diese werden geordnet, dedupliziert und aufbereitet. Dann läuft OpenCode zur Analyse drüber und das Ergebnis wandert als Issue nach GitHub. Von dort aus liegt die Entscheidungsgewalt wieder bei mir. Vergebe ich das Freigabe-Label, übernimmt der zweite Workflow und lässt die KI den Fix umsetzen.

Der grobe Ablauf sieht so aus:

1. Home Assistant schreibt einen Fehler in sein Protokoll.
2. Der Log Analyzer lädt im nächsten Durchlauf alle Fehler über SSH.
3. Die Zeilen werden zerlegt, dedupliziert und mit einem Fingerprint versehen.
4. Rauschen und zu seltene Warnings werden herausgefiltert.
5. Eine KI-Triage entscheidet, ob sich für den Fehler überhaupt ein Issue lohnt.
6. Es wird geprüft, ob der Fehler auf GitHub schon existiert. Falls nein, analysiert OpenCode ihn und erstellt ein Issue.
7. Ich werde benachrichtigt, prüfe das Issue und vergebe bei Bedarf das Label ai:ready.
8. Der Self-Healer greift sich das Issue, lässt die KI den Fix umsetzen, dokumentiert das Ergebnis und setzt das passende Status-Label.

3. Voraussetzungen

Bevor wir bauen, gehen wir die Voraussetzungen durch. Wenn du sie sicher abhakst, sparst du dir später einige Rückschritte.

- **Eine laufende n8n-Instanz.** Ob self-hosted oder in der n8n-Cloud, spielt keine Rolle. Wichtig ist, dass n8n deine anderen Systeme (Home Assistant, OpenCode-Host, GitHub) über das Netzwerk erreichen kann.
- **Home Assistant mit SSH-Zugang.** Es muss möglich sein, sich per SSH zu verbinden und dort ha core logs auszuführen. In der Praxis läuft das über das offizielle „Advanced SSH & Web Terminal"-Add-on oder einen vergleichbaren Zugang.
- **Ein Host mit OpenCode.** OpenCode muss installiert und über SSH erreichbar sein. Die eigentliche Voraussetzung ist ein funktionierender Home-Assistant-MCP in OpenCode plus ein hinterlegtes KI-Modell (bei mir ein OpenAI-Modell). Ob du dafür den Standard-Agenten nutzt oder – wie ich – einen eigenen Agenten anlegst, bleibt dir überlassen (mehr dazu in Kapitel 4.3).
- **Ein privates GitHub-Repository.** Hier landen die Issues. Ein privates Repo ist wichtig, da Logauszüge sensible Informationen enthalten können.
- **Ein OpenAI-API-Zugang.** Für die schnelle Triage-Klassifizierung im ersten Workflow. Alternativ lässt sich hier auch ein anderes Modell einsetzen.
- **Optional: ein Telegram-Bot.** Für die Benachrichtigung über neue Issues. Rein optional – der Workflow funktioniert auch ohne.

> **Backups sind Pflicht**
> Dieses System kann später eigenständig Änderungen an deinem Home Assistant vornehmen. Richte deshalb – bevor du startest – automatische, regelmäßige Backups ein und stelle sicher, dass du sie im Ernstfall auch zurückspielen kannst. Ohne sauberes Backup solltest du die zweite Stufe (den Self-Healer) gar nicht erst scharf schalten.